VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Unity 3D

Unity 3D

2014

Jan Wlosok

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Jan Wlosok**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Unity 3D

Unity 3D

Zásady pro vypracování:

Seznamte se s profesionálním vývojovým prostředím Unity 3D, popište jeho základní možnosti (např. možnosti networkingu, možnosti webového rozhraní, porovnání výkonu ve webovém prohlížeči vs. standalone, možnosti importu 3D modelů a animací z různých grafických aplikací).

1. Nastudujte vývojové prostředí Unity 3D.
2. Zaměřte se primárně na kolize objektů, fyziku, síťovou komunikaci, hru více hráču apod.).
3. Demonstrujte možnosti použití programu 3D Unity na "Virtuální autoškole".
4. Popište a otestujte postup načítání a úpravy modelu pro použití v Unity.
5. Praktické výsledky pečlivě teoreticky popište pro další využití.

Seznam doporučené odborné literatury:

Podle pokynů vedoucího bakalářské práce.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Martin Němec, Ph.D.**

Datum zadání: 16.11.2012

Datum odevzdání: 07.05.2014

VYSOKÁ ŠKOLA BÁŇSKÁ-TECHNICKÁ UNIVERZITA * Fakulta elektrotechniky a informatiky * OSTRAVA *

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 7. května 2014 .....

Rád bych na tomto místě poděkoval vedoucímu práce Ing. Martinu Němcovi, Ph.D. za cenné rady a připomínky při tvorbě této bakalářské práce.

Abstrakt

Tato bakalářská práce se zabývá herním enginem Unity 3D a jeho použitím v praxi. V první části jsou popsány základy herních enginů a jejich srovnání. Dále je zde možno najít popis hlavních částí herního enginu Unity 3D. Pro demonstraci jednotlivých částí Unity 3D byla vytvořená aplikace v podobě virtuální autoškoly. Tato aplikace je podrobně popsána v druhé části této práce.

Klíčová slova: Herní engine, Unity 3D, 3D grafika, Virtuální autoškola, Blender, Unity Script, Editor, Fyzika

Abstract

This bachelor thesis deals with Unity 3D game engine and its use in practice. The first section describes basics of game engine and its comparison. Furthermore, it is possible to find a description of the main parts of Unity 3D. To demonstrate the various parts of Unity 3D the application in the form of virtual driving school was created. This application is described in detail in the second part of this work.

Keywords: Game engine, Unity 3D, 3D graphics, Virtual driving school, Blender, Unity Script, Editor, Physics

Seznam použitých zkratek a symbolů

2D	–	Dvojrozměrný
3D	–	Trojrozměrný
CPU	–	Central Processing Unit
FPS	–	Frames Per Second / First Person Shooter
GPU	–	Graphics Processing Unit
GUI	–	Graphic User Interface
LOD	–	Level of Detail
RPC	–	Remote Procedure Call
RPG	–	Role Playing Game

Obsah

Seznam tabulek

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Herní enginy se za posledních deset let změnily k nepoznání. Množství nástrojů, vylepšení a technologických novinek je obrovské. A to si žádalo přehodnocení filozofie vývoje her, kde místo tvorby vlastního enginu, se začaly čím dál tím více používat enginy již zaběhnuté a ověřené. Tím se urychlil a v mnoha ohledech i usnadnil proces vývoje her. Zároveň se otevřely nové možnosti pro nezávislé vývojáře.

S nástupem digitální distribuce a později i chytrých telefonů se někteří výrobci herních enginů rozhodli pro změnu business modelu. Se záměrem prodat více licencí za menší cenu se tak otevřely možnosti pro nezávislé vývojáře. Herní enginy se ale staly nástrojem i pro ne herní projekty. Architektonické vizualizace, interaktivní prezentace nebo simulační aplikace. To je jen pár příkladů, kde našly herní enginy uplatnění.

Tato práce se zabývá herním enginem Unity 3D a jeho praktickým využitím. Nejdříve se zabývá obecnými principy herních enginů a jejich srovnáním. Dále je zde již konkrétní popis možností a nástrojů herního enginu Unity 3D. Další kapitoly se pak zabývají již praktickým využitím v podobě aplikace virtuální autoškoly. Zde jsou rozebrány jednotlivé kroky od návrhu přes detailní popis tvorby jednotlivých částí až po optimalizaci výsledné aplikace.

V závěru jsou zhodnoceny dosažené cíle a popsány případné nedostatky. Zároveň jsou zde zmíněny možnosti, jak by se dala aplikace vylepšit.

2 Herní engine

Herní engine tvoří jádro hry, se základními funkcemi společnými pro většinu her jako rendering, fyzika, zvuk, skriptování, animace, networking, atd. Tuto funkcionalitu lze využívat pomocí nástrojů dodaných s enginem. Nejčastěji se jedná o univerzální editor, ve kterém lze tvořit úrovně, materiály, pracovat s grafikou nebo vytvářet logiku pro hru.

Hlavním principem herních enginů je usnadnit práci herním vývojářům. Jelikož herní engine již obsahuje tu nejzákladnější funkcionalitu, dovoluje vývojářům soustředit se na samotnou tvorbu logiky a prostředí hry. Tím se proces výrazně usnadňuje a hlavně urychluje. Zároveň je ale u některých enginů možné získat přístup ke zdrojovým kódům, a pokud je to nutné, je možné upravit engine pro konkrétní účely.

Výhodou herních enginů je i přístupnost pro vývojáře s nižší znalostí programování. Jelikož editor obsahuje sadu nástrojů pro tvorbu a úpravu jednotlivých částí hry, není třeba zasahovat do samotného kódu hry. Ovládání editoru je v mnoha případech velmi podobné ovládání nejznámějších 3D modelovacích aplikací, jako je 3D Studio Max nebo Maya. Tím se usnadňuje práce grafikům, od kterých se dnes nečeká jen modelování a texturovaní, ale také otestování assetu v prostředí herního enginu, případně i tvorba materiálů.

Další velkou výhodou herních enginů, je jejich poměrně velká životnost. Herní enginy se vyvíjejí tak, aby je bylo možné používat co nejdéle. Jelikož se technologie posouvají velmi rychle dopředu, je tento cíl velmi obtížný. Řešením jsou časté updaty, které přidávají do enginů nejnovější technologie. Některé enginy se tak používají až neuvěřitelných deset let (Unreal Engine 3, Source Engine).

Jedním z důvodů, proč se začaly herní enginy používat ve velkém, byla nutnost multiplatformního vývoje. Herní enginy jsou navrženy tak, aby bylo předělání hry z jedné platformy na druhou co nejjednodušší. V posledních několika letech se důraz na multiplatformnost enginů ještě zvětšil. Důvodem byl nástup chytrých telefonů, tabletů a her ve webových prohlížečích.

2.1 Srovnání herních enginů

I přesto, že je daleko výhodnější využít již jeden z mnoha dostupných a ověřených herních enginů, některá studia si stále vytvářejí vlastní herní enginy. Jedním z hlavních důvodů je tvorba velmi neobvyklé hry nebo hry s velmi specifickými technickými požadavky. V takovém případě má tvorba vlastního enginu, přizpůsobeného potřebám vývojářů, smysl.

Běžnější praxí je využít jeden z dostupných enginů. V takovém případě je třeba se důkladně zamyslet, jakou hru se vytváří. Fakt, že se dá ve většině populárních enginů vytvářet různé typy her, neznamená, že je pro to každý z těchto enginů stejně vhodný. Proto je třeba prozkoumat různé možnosti jednotlivých herních enginů.

Mezi hlavní faktory, které je třeba vzít v potaz, při výběru enginu, je typ enginu, cena a podporované platformy. Zároveň je dobré zjistit, jaké hry již byly s tímto enginem vytvořeny. Srovnání nejpopulárnějších volně dostupných enginů: 1 .

Název enginu	Skriptování	Platformy	Cena	Hry
Unreal engine 3	Unrealscript, Kismet	Windows, Linux, Mac OS X, Xbox 360, PS3, PS Vita, Wii U, Andorid, iOS, Flash, HTML5	Individuální	Bioshock Infinite, Dishonored, Mass Effect 1-3, Remember Me
Unreal Engine 4	C++, Blueprint	Windows, Linux, Mac OS X, Xbox One, PS4, iOS, Andorid, HTML5	$19/měsíc + 5%	Daylight, Fortnite
CryEngine	Lua	Windows, Linux, Xbox360, Xbox One, PS3, PS4, Wii U	$9.99/měsíc bez zdrojových kódů; Cena se drojovými kódy - individuální	Ryse: Son of Rome, Evolve, Armored Warfare
Unity 3D	Unityscript, C#, Boo	Windows, Linux, Mac OS X, Xbox 360, Xbox One, PS3, PS4, PS Vita, Wii, Wii U, iOS, Andorid, Windows Phone, BlackBerry, Flash, HTML5, Unity Web Player	$1500 nebo 75/měsíc obě varianty bez zdrojových kódů	Wasteland 2, Rust, Dead Trigger 2, The Room, République, Pillars of Eternity, Gone Home, Deus Ex"The Fall
Source Engine	Squirrel, Lua	Windows, Linx, Mac OS X, PS3, Xbox, Xbox360	Individuální	Counter Strike: Source, Team Fortress 2, Left 4 Dead, Half-Life 2, Portal, Portal 2, Dota 2, Titanfall
Havok Vision	Havok Script	Windows, PS3, PS Vita, Xbox360, Wii, Wii U, iOS, Andorid	Individuální	The Settlers 7: Paths to a Kingdom, Stronghold 3, Arcania: Gothic 4, Orcs Must Die! 2

Tabulka 1: Srovnání herních enginů

Z těchto enginů se tři vyjímají. Unreal Engine, CryEngine a Unity 3D totiž míří na podobnou cílovou skupinu a nabízí podobné řešení za podobnou cenu. Ještě nedávno tomu tak nebylo. CryEngine mířil na vysoce graficky realistické hry, Unreal Engine se specializoval na hry s uzavřenými úrovněmi a Unity 3D byl primárně nástroj nezávislých vývojářů. Tyto rozdíly se však smazaly po změně byznys modelu u Unreal Enginu a poté i u CryEnginu. Nyní jsou dostupné nejen pro velká studia, ale i pro nezávislé vývojáře. Ti si nyní mohou vybrat mezi třemi velmi kvalitními enginy.

2.2 Unity 3D

Unity 3D (dále Unity) je multiplatformní herní engine vyvíjený společností Unity Technologies. Používá se pro vývoj počítačových, konzolových, mobilních, ale i webových her. Nabízí kompletní balení nástrojů nutných pro tvorbu kvalitních her a interaktivních aplikací.

První verze, původně vytvořená pro vývoj OS-X her, vydána v roce 2005 se rychle rozrostla do jednoho z nejlepších multiplatformních balíků nástrojů pro vývoj her. Dnes se již jedná o nejpoužívanější engine na světě. Jeho primárním polem působnosti je mobilní trh. Obrovské množství Android i iOS her je vytvořeno právě v Unity. To ovšem neznamená, že zaostává na ostatních platformách. V několika posledních letech začíná být Unity velmi populární i mezi vývojáři počítačových her.

Důvodů, proč je Unity tak populární, je mnoho, ale hlavním důvodem je všestrannost. Rozsah formátů, které Unity umožňuje importovat, nemá mezi herními enginy obdobu. I způsob, jakým se s jednotlivými assety v Unity pracuje, je velmi intuitivní. Proto je Unity velmi populární i mezi vývojáři více zaměřenými na grafiku.

2.3 Unreal Engine

Unreal engine je herní engine vyvíjený společnosti Epic Games. Je to jeden z nejpoužívanějších AAA enginů v průmyslu.

Unreal Engine byl vydaný v roce 1996. V roce 1998 poprvé použit ve hře Unreal. I přesto že byl vytvořen primárně pro FPS hry, stal se popuární i v jiných žánrech. Jak rostla popularita, tak se rozšiřovaly možnosti Unreal Enginu. Netrvalo dlouho a stal se jedním z nejoblíbenějších enginů.

Unreal Engine obsahuje velmi kvalitní editor. Ten umožňuje vývojářům jak velmi rychlou tvorbu prototypů, tak i tvorbu komplexních úrovní. Obsahuje také velmi pokročilý editor materiálů, nástroj pro vizuální skriptování (Kismet/Blueprint), editor vizuálních a částicových efektů (Cascade), a mnoho dalších nástrojů.

Nyní ve své čtvrté verzi nabízí jeden z nejkvalitnějších vývojářských nástrojů na trhu za neuvěřitelně přijatelnou cenu. Tím se stává výborným nástrojem nejen pro velká studia, ale i pro nezávislé nývojáře.

2.4 CryEngine

Cryengine je herní engine vyvíjený německou společností Crytek. Jedná se o kompletní balíček nástrojů pro vývoj her na různé platformy.

První verze CryEnginu byla oznámená v roce 2002. O rok později bylo možné vidět první hru využívající CryEngine a tou byl Far Cry. Hra získala mnoho ocenění a zaujala i velmi kvalitním grafickým zpracováním, možným právě díky CryEnginu. I přesto, že je CryEngine kompletní balík velmi kvalitních nástrojů, nejvíce zaujal jeho renderer. CryEgine má bezesporu jeden z nekvalitnějších rendererů v herním průmyslu.

CryEngine je herní engine vhodný pro tvorbu her s obrovskými lokacemi. K tomu je přizpůsobený i jeho editor a nabízí například výborný nástroj pro tvorbu terénu, nástroje pro tvorbu řek a cest nebo systém procedurálního umisťování assetů. Samozřejmě kromě těchto specifických nástrojů obsahuje i klasické nástroje jako editor materiálů nebo logický editor.

Od roku 2002 se CryEngine dostal do své čtvrté verze pojmenované jednoduše CryEngine. Stal se zároveň jedním z nejkvalitnějších enginů.

3 Unity 3D - nástroje

Unity je kompletní balení nástrojů pro tvorbu her. Jednou z hlavních předností tohoto enginu je velká komunita vývojářů. Toho využili i samotní vývojáři Unity a vytvořili Asset Store. Asset Store je obchod, kde mohou vývojáři nakupovat 3D modely, textury, nástroje, skripty, atd., a tím urychlit vývoj nebo alespoň prototyp hry. Mohlo by se zdát, že vytvořit hru je nyní velmi jednoduché. S dostupnou grafikou a velkým množstvím ukázkových skriptů, to tak vypadá. Opak je ale pravdou. Možnost nakoupit grafiku urychlí proces vývoje a ukázkové skripty dovolí vývojářům získat přehled o jednotlivých mechanikách. Ovšem hlavní části vývoje hry je tvorba samotné hratelnosti a vytváření úrovní. Tento proces je velmi časově náročný a vyžaduje znalost mnoha nástrojů potřebných pro úspěšné dokončení a vyladění výsledné hry.

3.1 Editor

Pro vývojáře je hlavní částí editor 1 . Ten obsahuje panely a okna pro úpravu vlastností jednotlivých objektů, používání různých nástrojů nebo tvorbu samotné scény. Výhodou editoru je možnost spuštění rozpracované hry přímo v něm. Tím je možné ladit hru přímo za běhu, což je pro vývojáře velkou výhodou. Editor lze samozřejmě přizpůsobit dle libosti a tyto nastavení poté uložit. Je tak možné vytvořit několik layoutů editoru pro různé činnosti a libovolně mezi nimi přepínat. Z editoru se lze i jednoduše přepnout na dokumentaci Unity a to jak online tak i offline dokumentaci, která se instaluje společně s Unity.

3.2 Asset Store

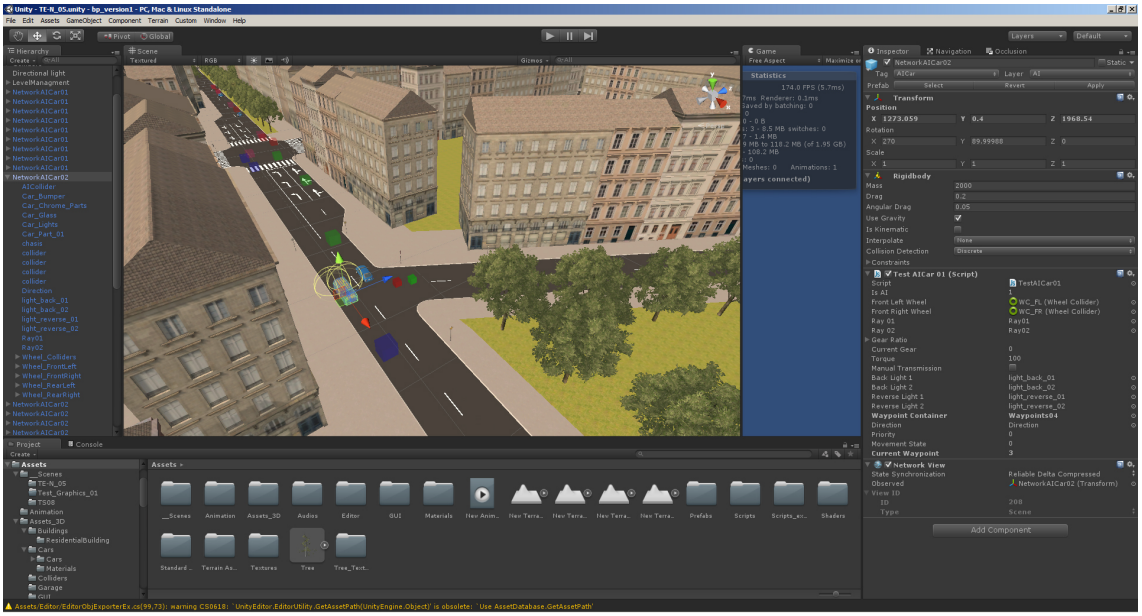
Asset store je internetový obchod, kde mohou vývojáři nakupovat nejen obsah pro své hry, ale také nástroje pro rozšíření Unity nebo dokonce celé projekty konkrétních her. Tím se urychluje proces vývoje. Zároveň mají možnost tvořit kvalitní hry i ne graficky znalí vývojáři, kteří si mohou modely, textury a animace nakoupit na Asset storu.

Unity zaobaluje většinu svých funkcí do nástrojů přístupných pomocí uživatelského rozhraní. Obsahuje ale rovněž takové funkce, které toto rozhraní nemají. Tím se stává práce s těmito funkcemi obtížná a často zbytečně časově náročná. Unity ovšem dovoluje vývojářům tvořit vlastní nástroje a právě takto lze tyto problémy vyřešit. Tyto nástroje lze koupit na Asset storu. Je tak možné přidat do Unity editor pro tvorbu materiálů, vizuální editor logiky, nástroj pro vertex painting nebo třeba nástroj pro rychlejší tvorbu prototypů úrovní.

Asset store ale nabízí i nástroje vytvořeny pro konkrétní typ hry. Lze tam nalézt nástroje pro tvorbu RPG her, dialogů ve hře, závodních her, apod.

3.3 Skriptování

V Unity se pro vytvoření jakýchkoliv mechanismů a logiky používá skriptování. Od enginů jako je Unreal Engine nebo CryEngine se v tomto ohledu Unity liší. Tam se totiž



Obrázek 1: Unity 3D - Editor

pro velkou část tvorby logiky hry používá vizuálního skriptování. Jedná se o editor, kde je možné pokládat a spojovat jednotlivé logické bloky. Lze tak v mnoha případech vytvořit i složité objekty bez nutnosti psaní kódu. Unity takový nástroj postrádá, lze jej však dokoupit na Asset Storu.

I přesto, že Unity nedisponuje vizuálním skriptováním, je tvorba jednotlivých částí logiky velmi plynulá. V Unity je možno skriptovat jedním ze tří jazyků. Těmito jazyky jsou UnityScript, CSharp a Boo. Je dokonce možné v rámci jednoho projektu, používat skripty napsané různými jazyky, to se ovšem pro lepší organizaci projektu, nedoporučuje. Mezi skriptovacími jazyky není prakticky žádný výkonový rozdíl. Volba jazyka je závislá spíše na preferenci vývojáře.

Samotné skriptovací jazyky jsou velmi dobře zdokumentované. Dokumentace je dostupná jak online, tak i lokálně na počítači, kde je Unity nainstalováno. Popis jednotlivých tříd a funkcí je doplněn o ukázky kódu pro lepší pochopení.

3.4 Objekty, komponenty a prefaby

V Unity se k sestavení komplexnějších objektů používají tzv. game objecty. Ty využívají dědičnosti, proto je možné přidat jeden game object na druhý. Přidaný game object budě dědit některé parametry svého rodičovského objektu. Vlastnosti game objectu jsou definovány komponentami. Ty lze na game objecty libovolně přidávat. Objekt sestavený z jednoho nebo více game objectů a s přidanými různými komponentami lze pak uložit jako prefab.

3.4.1 Game object

V Unity je každý objekt obsažený v scéně game object. Chování jednotlivých game objectů určují tzv. komponenty. Nejnižší forma game objectu je tzv. empty object, neboli prázdný objekt. Ten obsahuje pouze základní komponentu transform, která udává jeho pozici, rotaci a velikost.

3.4.2 Komponenty

Komponenty určují jednotlivé vlastnosti game objectů. Každý game object obsahuje neodstranitelnou komponentu transform, která udává jeho pozici, rotaci a velikost. Veškeré vlastnosti přidané na game object jsou tvořeny komponentami. Například objekt kamery je ve výsledku pouze prázdný game object s přidanou komponentou kamera. Mezi komponenty patří i skripty připojované na game objecty.

3.4.3 Prefab

Prefab je jeden nebo více game objectů se specifickým nastavením spojených dohromady pro snadnější použití ve hře. Výborným příkladem je například automobil, který se skládá z 3D assetu auta, oddělených kol, wheel colliderů, colideru samotného auta, kamery, skriptu pro ovládání, atd. Nebylo by praktické tento složitý asset vytvářet pokaždé, když potřebujeme jeho instanci ve scéně. Proto se vytvoří jen jednou a uloží se jako prefab, který lze jednoduše přidat do scény. Výhodou prefabu je, že se dají globálně upravovat. Pokud upravíme jednu instanci, můžeme aplikovat úpravy na všechny ostatní. Zároveň ale můžeme mít několik instancí téhož prefabu a každé nastavit rozdílné parametry.

4 Návrh demonstrační aplikace

Praktickou částí této bakalářské práce bylo vytvoření aplikace v podobě virtuální autoškoly. Hlavním cílem této aplikace není fyzikální simulátor řízení, ale ověření znalostí předpisů o provozu na pozemních komunikacích. Jízdou v otevřeném světě mezi ostatními účastníky provozu je tak možné otestovat teoretické znalosti při rozhodování v reálném čase. Účastníky provozu tvoří jak umělá inteligence, tak i ostatní hráči, kteří se mohou do hry připojit. Tím vzniká určitý faktor náhodnosti.

4.1 Koncept

Jelikož bylo třeba pro tuto aplikaci vytvořit 3D grafiku, 2D grafiku, herní objekty, skripty, atd., a všechny tyto části spolu souvisí, bylo zapotřebí určit cíle aplikace a naplánovat jednotlivé mechanismy. V opačném případě by mohlo být promrháno mnoho hodin tvorbou nefungujících nebo zbytečných částí. Proto byly hned na začátku určeny hlavní části a mechanismy, které bylo zapotřebí udělat. Díky tomuto plánování bylo možné zjistit, jaké nástroje budou zapotřebí k tvorbě této aplikace, případně jaké nástroje chybí a přehodnotit tak vývoj částí, kterých se to týká.

4.2 Mapa

Již od začátku bylo jasné, že bude zapotřebí vytvořit otevřený svět. Bylo tedy zapotřebí navrhnout mapu. Ta má rozměry 3 km na 1,5 km. Takto velká mapa s sebou přináší spoustu problémů. Jedním z nich je počet objektů, které je třeba vytvořit, pro zaplnění mapy. V tomto případě byl tento problém vyřešen vytvořením modulárních assetů, kdy se mohla jedna budova rozšířit, zúžit nebo mohlo být přidáno více pater.

Dalším problémem byla tvorba terénu a sítě silnic. Unity totiž nedisponuje žádným nástrojem pro práci s křivkami nebo tvorbou silnic. Jde sice koupit nástroj pro pokládání silnic, který navíc reaguje na terén, ten je ale vhodný spíše pro závodní hry, jelikož nedokáže pracovat s křižovatkami. Proto byla síť cest vytvořena ve 3D aplikaci. I přesto, že Unity má zabudovaný nástroj pro tvorbu terénu, byl pro větší přesnost vytvořen terén také ve 3D aplikaci.

S takto velkou mapou je třeba řešit i optimalizaci vykreslování. Proto byly použity pro modely budov tzv. LOD. LOD umožňuje přepínat modely s různou hustotou detailů podle toho, jak daleko je od konkrétního modelu hráč. Další optimalizační technikou je occlusion culling. To dovoluje před vypočítat viditelnost objektů a vykreslovat objekty pouze reálně viditelné hráčem.

5 Práce s grafikou

Jednou z hlavních předností herních enginů, je zpracování grafiky. Vývojář se nemusí zabývat tím, jak vykreslovat grafiku, jelikož herní engine již obsahuje renderer, který se o to postará. Je ale třeba určit, jakým způsobem se grafika vykresluje. Výsledek je závislý na tom, jak je model vymodelovaný a natexturovaný, to je ovšem práci grafika. Na straně enginu je nejdůležitější částí nastavit materiál. Ten určuje jednotlivé vlastnosti objektu, jako je barva, spekulární složka, průhlednost, atd.

5.1 3D modelování

Na začátku tohoto procesu je zapotřebí 3D model nebo animaci vytvořit. K tomuto účelu se používají 3D modelovací aplikace. Mezi tyto aplikace patří například 3D Studio Max, Maya, Blender, Modo, 4D Cinema, a mnoho dalších. Při modelování assetů pro hry je zapotřebí mít na paměti několik věcí. První z nich je počet polygonů neboli trojúhelníků. I přesto, že v dnešní době není problém vykreslovat scény s několika miliony polygonů, je dobré modelovat úsporně. Další důležitý faktor, který určuje dobře vymodelovaný model, je topologie. Topologie 3D modelu je způsob, jakým je rozložená síť bodů. Čistá topologie obsahuje smysluplně rozložené smyčky bodů. Jedním z posledních kroků, které je třeba provést velmi pečlivě, je uv mapování. V podstatě se jedná o způsob, jak rozřezat 3D model a převést ho do 2D souřadnic. Tento krok je důležitý pro správné mapování textur.

Po vymodelování objektu je zapotřebí udělat ještě jeden důležitý krok. Tím je export modelu do vhodného formátu. Většina dnešních enginů používá formát Autodesk fbx (.fbx), jsou ovšem i takové, které používají jiné formáty a často je zapotřebí nainstalovat speciální exporter. Unity dokáže zpracovat řadu formátů, ale ve výsledku vše převádí do fbx. Pro předvídatelnější výsledky je proto lepší modely exportovat přímo do fbx a ty poté naimportovat. Při exportu je totiž možné nastavit co vše se má do souboru přibalit. Například model budovy, který v Unity scéně reprezentuje statický model, je zbytečné exportovat i s animacemi. Dalším parametrem, který může být při exportu užitečný, je nastavení 3D os. Některé enginy totiž mohou mít osy obrácené jinak, což by po importování vedlo k nesprávně odrotovanému objektu.

Pro modelování assetů do této aplikace byl použit program Blender. Oproti velkým komerčním aplikacím, jako je 3D Studio Max nebo Maya má blender jednu velkou výhodu. Je distribuován zdarma. I přesto obsahuje většinu nástrojů, které lze nalézt v komerčních aplikacích.

5.2 Texturování modelů

Textůrování modelů je pravděpodobně ještě důležitější krok tvorby grafiky, než samotné modelování. Důvodem jsou daleko větší limitace. Grafické karty zvládnou vykreslovat velké množství polygonů, ale mají velmi omezenou velikost paměti. A právě paměť grafické karty je limitace, kterou je třeba mít na mysli při tvorbě textur. Proto je třeba se při uv mapování snažit vměstnat jednotlivé části modelu co nejtěsněji k sobě.

Textůry se ve většině případů vytváří jako čtvercový obrázek. Co se týče velikostí, je dobré využívat mocnin čísla 2 (256, 512, 1024, 2048). Enginy většinou obsahují nástroje pro zmenšování textur, kdy je možné vytvořit texturu 2048*2048 pixelů, ale ve výsledku nastavit pouze velikost 1024*1024 pixelů. Při použití mocnin čísla dva je toto zmenšování nejefektivnější.

Proces vytváření textur může probíhat různými způsoby. Nejjednodušším způsobem je použít foto texturu namapovanou přímo na objekt. Tento proces je vhodný například pro terény, nebo jednoduché geometrické tvary, pro většinu ostatních modelů je třeba model uv mapovat a poté texturu ručně vyrobit. Ruční tvorba textur probíhá ve 2D programu, kde se pomocí různých technik míchají a upravují foto textury. Pro stylizované objekty se textury ve 2D programech kreslí. Další možností je kreslení textur přímo na 3D objekty. Zde se využívá jak kreslících schopností grafika, tak i chytrého mapování různých fotografií na 3D model.

Poměrně novým způsobem texturování je texturování pomocí procedurálních nástrojů, jako je Substance Designer nebo dDo. Tyto nástroje využívají ostatních textur, jako je normal mapa, cavity mapa, selection mapa, atd. Pomocí těchto textur dokážou určit, kde na objektu se nachází hrany, drážky, apod. a jak jsou ostré. Díky těmto informacím lze nanášet vrstvy různých efektů, které ve výsledku mohou vytvořit například komplexní povrch rezavějícího kovu. Výhodou těchto programů je automatické generování ostatních map, jako je specular mapa, gloss mapa nebo normal mapa. Zároveň lze jednotlivé postupy uložit a použít znovu na jiný model. Tím se ušetří obrovské množství času.

Pro tuto aplikaci byl použit Photoshop CS3. Photoshop je leaderem na poli 2D editorů. Obsahuje množství nástrojů, které nejen dovolují vytvářet kvalitní textury, ale také urychlují celý proces. Tato bakalářská práce se ovšem nesoustředí na grafiku, nýbrž na funkčnost. Proto nebyla texturovaní přikládaná taková důležitost a pro většinu modelů byla vytvořená pouze difuzní mapa.

5.3 Import assetů

Důležitou části práce s grafikou je správný import assetů. Unity disponuje velmi jednoduchým způsobem importu, kdy stačí přetáhnout assety do projektu a ty se automaticky naimportují. To ovšem neznamená, že tím je proces importu hotov. U jednotlivých assetů je zapotřebí nastavit správné hodnoty.

U tohoto projektu byl použit pro modelování Blender. Modely vyexportované z Blenderu jsou po importu do Unity s nesprávnou velikostí, proto je zapotřebí změnit při importu Scale z 0.01 na 1. Zároveň byly modely vytvořeny bez materiálů, jelikož se materiály vytvářely později přímo v Unity. Proto je dobré neimportovat materiály společně s modelem. Toho lze docílit odškrtnutím Import materials. U statických objektů, jako jsou budovy, značky, chodníky, atd. byla zaškrtnuta volba Generate lightmap UV. Unity tak vygeneruje druhou uv mapu, která se později použije pro lightmapy.

U importování textur není třeba mnoho změn. V případě této práce byla u některých textur změněná velikost z 2048*2048 pixelů na 1024*1024 pixelů.

5.4 Materiály a shadery

Materiály a shadery jsou v Unity úzce spojeny. Shader obsahuje kód, který určuje, jaké vlastnosti má používat. Materiál dovoluje tyto vlastnosti upravovat v editoru.

Existují tři způsoby, jak psát shadery v Unity:

- Fixed Function Shaders – tato metoda je nejrychlejší a proto se používá převážně tam, kde je hardwarový výkon problém. Často se také tímto způsobem píšou alternativy pro komplexnější surface shadery. Důvodem je nutnost spuštění na starším hardware, který by komplexní shadery nezvládal.Fixed Function Shaders se píšou v jazyce ShaderLab, podobný NVIDIA CgFX.
- Vertex/Fragment Shaders – často používané pro shadery, které nepotřebují interagovat se světlem nebo shadery, které jsou velmi netradiční a nelze tak dosáhnout požadovaných výsledků v Surface Shaders. Psaní shaderu tímto způsobem dává vývojářům největší svobodu. Zároveň se ale jedná o složitější metodu, která vyžaduje daleko více kódu. Tyto shadery se píšou v jazyce Cg/HLSL.
- Surface Shaders – Surface shaders je nejlepší možností pro psaní shaderů, pokud má shader interagovat se světlem. Tento způsob psaní shaderů je nejlepší pro ne příliš složité nebo specificky zaměřené shadery. Není zde zapotřebí psát tolik kódu jako u Vertex/Fragment. Unity automaticky kompiluje tento kód do klasických vertex/fragment shaderů, které lze posléze dle libosti editovat. Každý Surface shader je možné napsat přímo jako vertex/fragment shader. To ovšem v opačném případě nemusí platit. Surface Shader je způsob psaní shaderu na vyšší úrovni, kde se sice zjednoduší práce, ale zároveň se zmenší kontrola nad shaderem. Tyto shadery se pišou v jazyce Cg/HLSL.

Nevýhodou systému shaderů v Unity je nutnost znalosti jazyka Cg/HLSL. Unity sice obsahuje sadu shaderů pro běžné použití, ale jakmile je zapotřebí vytvořit složitější materiál, je znalost psaní shaderů nezbytná. Enginy jako je Unreal Engine nebo CryEngine využívají pro tvorbu materiálů uzlový editor (node-based editor). Ten dovoluje vývojářům vytvořit materiál na míru pomocí vizuálního editoru využívajícího funkční uzly, které lze různě spojovat a dosáhnout tak velmi specifických či komplexních výsledků.

5.5 Shadery vytvořené pro ukázkovou aplikaci

V této bakalářské práci byly kromě standardních shaderu použity i vlastní shadery. Ty byly vytvořeny jako sruface shadery. Jedná se o shadery vytvořeny pro urychlení práce s grafikou. Primárně se jedná o různé shadery využívající vertex colors. Unity nedisponuje žádným podobným shaderem.

Jedním z těchto shaderů byl shader pro maskování tří různých textur podle vertex colors 3D modelu 1. Textury byly maskovány jednotlivými kanály RGB, kdy každý kanál určoval viditelnost jedné textury. Díky tomuto shaderu se urychlil proces texturovaní, jelikož stačilo rovnoměrně uv mapovat celý objekt, obarvit jednotlivé části modelu a materiálu přiřadit textury.

```
Shader "VertexColorShaders/RGBVertexColorSelect"
{
    Properties
    {
        _Color ("Main␣Color", Color) = (1,1,1,1)
        _MainTex ("Base␣(RGB)", 2D) = "white" {}
        _vertStrength("Vert␣Color␣Strength", Range(0.0, 2.0)) = 1.0
        _Texture2 ("Base2␣(RGB)", 2D) = "white" {}
        _Texture3 ("Base2␣(RGB)", 2D) = "white" {}
    }
    SubShader
    {
        Tags { "RenderType"="Opaque" }
        LOD 200

        CGPROGRAM
        #pragma surface surf BlinnPhong vertex:vert

        sampler2D _MainTex;
        sampler2D _Texture2;
        sampler2D _Texture3;
        fixed4 _Color;
        float _vertStrength;


        struct Input
        {
            float2 uv_MainTex;
            float3 vertColors;
        };

        void vert(inout appdata_full v, out Input o)
        {
            o.vertColors = v.color.rgb;
            o.uv_MainTex = v.texcoord;
        }

        void surf (Input IN, inout SurfaceOutput o)
        {
            fixed4 c = tex2D(_MainTex, IN.uv_MainTex) * _Color;
            fixed4 tex2 = tex2D(_Texture2, IN.uv_MainTex) * _Color;
            fixed4 tex3 = tex2D(_Texture3, IN.uv_MainTex) * _Color;
            tex2.rgb = lerp(tex2.rgb, c.rgb, IN.vertColors.g);
            tex3.rgb = lerp(tex2.rgb, tex3.rgb, IN.vertColors.r);
            o.Albedo = tex3.rgb;
            o.Alpha = tex3.a;
        }
        ENDCG
    }
    Fallback "VertexLit"
}
```

Výpis 1: Shader - maskování textur pomocí RGB kanálů vertex colors

6 Sestavování objektů a prefabů

Je dobrou praktikou tvořit ze všech objektů, umisťovaných do scény, prefaby. To dovoluje rychlou editaci mnoha objektů bez nutnosti jednotlivě objekty vyhledávat a měnit. 3D modely naimportovány do projektu se automaticky chovají jako prefab. Z ostatních objektů vytvořených pro tuto aplikaci byly vytvořeny prefaby.

6.1 Objekt hráče

Jedním z nejdůležitějších objektů této aplikace byl objekt hráče. V tomto případě se jedná o automobil. Ten se skládá z mnoha částí, často i na sobě závislých.

Nejedná se ovšem o obyčejné řiditelné vozidlo, jaké můžeme znát ze závodních her, ale jde přímo o objekt hráče. Veškeré mechanismy spojené přímo s konkrétním hráčem, jsou obsaženy v tomto objektu. Cílem tedy bylo vytvořit vozidlo, které jde řídit, ale zároveň detekuje hráčovy dopravní prohřešky.

6.1.1 Model auta

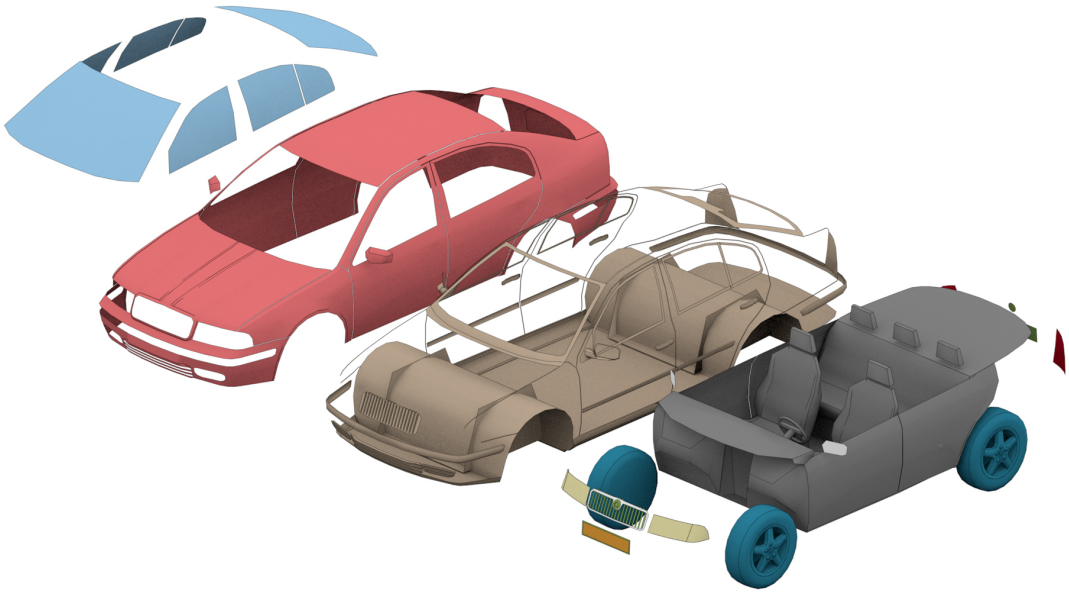
Model auta musel být před importováním upraven v 3D editoru. Bylo nutné sloučit částí stejných materiálů tak, aby se snížil výsledný počet draw callů v Unity 2 . I přesto se automobil skládal z velkého počtu objektů, proto byl vytvořen shader, který obarvoval jednotlivé části podle barev vertexů. Tímto řešením bylo možné sloučit části se stejným materiálem, ale jinou výslednou barvou. Dalším nutným krokem bylo oddělení kol od ostatních částí modelu a umístění origin bodu do středu kola. Tento krok je klíčový pro správné navázání na wheel collider v Unity.

6.1.2 Kolizní objekty a triggery

Aby mohl automobil kolidovat s ostatními objekty, bylo nutné vytvořit kolizní model. Ten byl vytvořený umístěním a změnou velikosti několika krychlí. Výsledný tvar přibližně kopíroval tvar auta. Je důležité, aby se jednotlivé krychle nikde nepřekrývaly. V opačném případě by se na tomto místě násobila hmotnost, což by vedlo k velmi nepřesné fyzice auta. U těchto krychlí se následně odstranila komponenta mesh renderer, jelikož je nežádoucí, aby se zobrazovaly.

Kromě kolizních objektů, obsahuje automobil i dva triggery 3 . Ty fungují jako spouštěče různých událostí. Oba triggery jsou vytvořené ze základní krychle, roztažené do potřebných rozměrů. Oba také mají vypnuté mesh renderery a v komponentě collider mají zaškrtnutý parametr trigger.

Trigger umístěný vepředu auta, je používán pro detekci kolize s ostatními auty. Zde se přistoupilo na kompromis a předpokládalo se, že hráč, který do někoho naboural, je viníkem. Tato metoda má samozřejmě nedostatky. Pokud by někdo na hráče nacouval, byl by neprávem označený za viníka. Ovšem je třeba si uvědomit, že v mnoha případech je velmi těžké programově určit viníka nehody. Navíc to není hlavním cílem aplikace.



Obrázek 2: Model rozdělený na části. Barvy určují jednotlivé části.

Druhým triggerem, je objekt použitý pro spouštění většiny ostatních detekcí prohřešků. Ten kopíruje velikost auta, aby byla detekce co nejpřesnější.

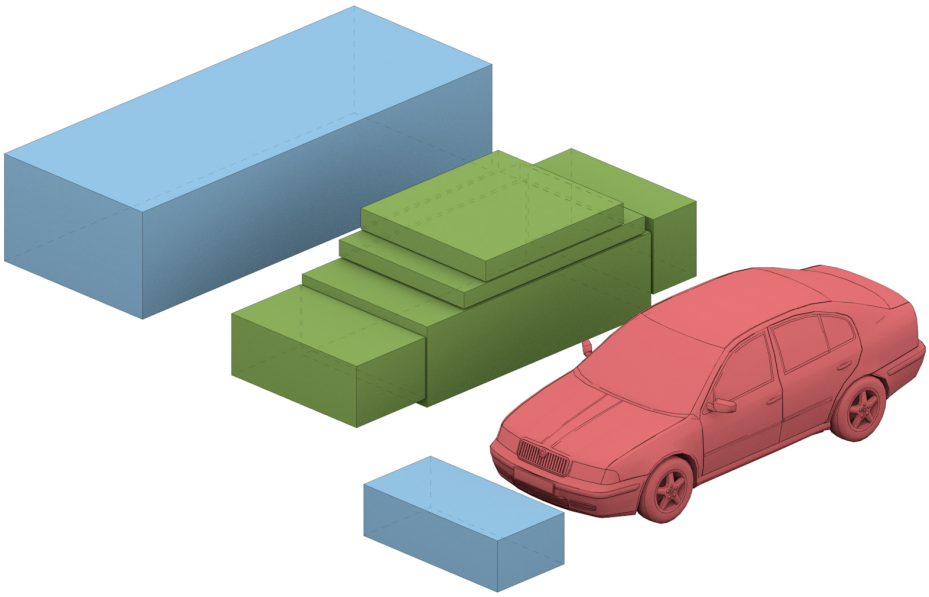
6.1.3 Ovládání auta

Pro tuto aplikaci byl vytvořen vlastní ovládací skript auta a to i přesto, že lze použít ovládací skript z ukázkových aplikací unity, či stáhnout některý ze zdarma dostupných na internetu. Jedním z důvodů byla nevhodnost většiny dostupných skriptů. Ty byly vytvořeny buď pro závodní hry, nebo hry s příliš reálnou fyziku, což znamenalo špatné ovládání na klávesnici.

Jelikož nebyla tato aplikace zamýšlená jako fyzikální simulátor řízení, byl vytvořen skript více blížící se arkádovému řízení. Zároveň bylo zapotřebí vzít v potaz, že nejčastěji používané ovládací zařízení bude klávesnice, která nedovoluje analogový vstup, a tudíž se s ní velmi špatně ovládají fyzikálně realistická vozidla. I přesto je ale možné ovládat vozidlo jak gamepadem, tak i volantem, jelikož byl ve skriptu použity pro ovládání vstup analogových os místo konkrétních tlačítek.

6.1.4 Detekce dopravních přestupků

Protože se dopravní přestupky detekují vždy pro konkrétního hráče, byl skript pro hlídání těchto přestupků také připojen na hlavní objekt hráče (v tomto případě automobil). Tento skript detekuje a zpracovává veškeré dopravní přestupky způsobené hráčem. Zároveň mění podle aktuálního dopravního přestupku text. Ten je totiž využíván GUI



Obrázek 3: Oddělené triggery(modré) a kolizní objekty(zelené).

skriptem pro zobrazení dopravního přestupku. Kromě detekce přestupku je zde také uchováván celkový počet bodů, který ubývá s každým dopravním přestupkem, ale při správné jízdě po určitou dobu, se začnou body znovu připisovat.

6.2 Umělá inteligence

Aby mapa nepůsobila prázdně, byla vytvořená umělá inteligence pro řízení automobilů. Hlavním cílem bylo vytvořit vozidlo, které dokáže jezdit po předem určené trase, respektuje dopravní pravidla a detekuje možnou kolizi s ostatními automobily.

6.2.1 Systém směrových bodů

Hlavním požadavkem bylo, aby auto řízené umělou inteligencí, dokázalo jezdit po předem určené trase. Nejjednodušší způsob, jak dosáhnout tohoto cíle by bylo animovat pohyb tohoto auta po křivce. Bohužel Unity nedisponuje žádným nástrojem pro tvorbu křivek. Lze ale využít jednoho z mnoha nástrojů třetích stran, dostupných na Unity Asset storu. I přesto, že tyto nástroje jsou vytvořeny právě pro pohyb objektů po křivce, nebylo toto řešení optimální.

Proto byl pro tento účel vyvinut systém využívající směrové body (waypoint). Základním principem tohoto systému, byla schopnost nasměrovat jízdu k dalšímu aktivnímu směrovému bodu. Proto bylo zapotřebí vyřešit otázku organizace směrových bodů a také samotné zpracování v ovládacím skriptu umělé inteligence.

Směrové body byly vytvořeny pomocí prázdných game objectů. Ty byly následně umístěny do jednoho prázdného game objectu, pro lepší organizaci. Tento objekt, obsahující sadu směrových bodů, je poté předán ovládacímu skriptu pro zpracování. Ovládací skript získá transformace jednotlivých bodů. Ty jsou poté využity k vypočtení úhlu mezi autem a následujícím směrovým bodem. Tento úhel se porovná s úhlem auta a tím se určí, na kterou stranu má auto zatáčet.

Tento systém ovšem nefungoval velmi dobře. Jízda aut, hlavně přes ostré zatáčky, byla příliš zběsilá a nebylo jednoduché samotnými směrovými body přesně určit cestu, kudy má auto jet. Proto bylo zapotřebí systém upravit pro lepší ladění pohybu aut. První úprava se týkala samotné jízdy aut přes zatáčky. Bylo zapotřebí měnit úhel zatáčení podle toho, jak ostrá je zatáčka. Pro tento účel byla vytvořena funkce, která vypočítá adekvátní úhel zatáčení podle úhlu zatáčky a rychlosti auta. Tato funkce zároveň sníží úhle zatáčení na velmi malou hodnotu, pokud se auto pohybuje velmi rychle. Tím se zabrání trhání auta při korekci směru.

Dalším problémem, který bylo nutné opravit, byla rychlost projetí zatáčkou. Tento problém byl vyřešen korekcí rychlosti podle úhlu a vzdálenosti k dalšímu směrovému bodu.

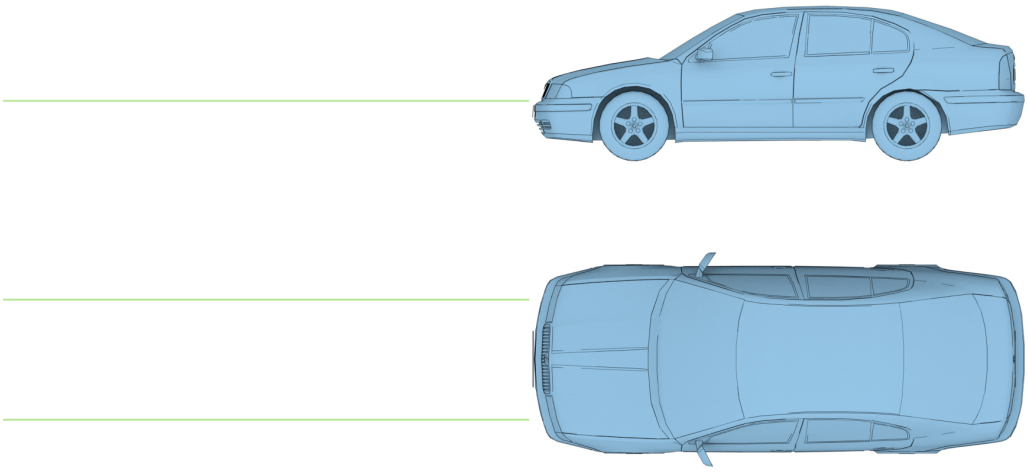
6.2.2 Detekce kolize

Důležitým prvkem této umělé inteligence byla i detekce ostatních účastníku provozu, pro vyhnutí se kolize s nimi. Pro tento účel byl nejdříve použit objekt krychle umístěný několik metrů před automobil. Tento objekt měl vypnutý rendering a fungoval jako trigger. Pokaždé, když začal protínat jiný objekt, odeslal se signál pro zastavení. Jak se ale později ukázalo, tato technika má mnoho nedostatků. Největším byl fakt, že neexistuje statická vzdálenost umístění tohoto objektu, která by fungovala ve všech případech. Pokud jelo auto rychle, nestačilo zastavit, jelikož byl objekt umístěn příliš blízko. Pokud byl ale objekt umístěn dále, mohl detekovat i automobily, se kterými by ve výsledku vůbec nekolidoval, například při zatáčení. Tento systém se proto ukázal jako nevhodný.

Nahradil jej systém využívající raycasting 2. Z přední části automobilu jsou vystřelovány dva paprsky směrem před automobil 4. Tyto paprsky detekují objekty, se kterými se protínají. Poté se porovná, zda se jedná o objekty se specifickým tagem, v tomto případě objekty automobilů umělé inteligence a hráče. Pokud ano, zahájí se proces brzdění. Výhodou tohoto systému je, že je možno paprsky měnit na základě různých informací. V tomto případě se mění délka paprsků podle rychlosti jízdy auta. Zároveň se paprsky zkracují na minimum, když auto projíždí ostrou zatáčkou. Tím se zabraňuje náhodnému zachycení ostatních vozidel.

```
raycast01 = Physics.Raycast (ray01.transform.position + Vector3(0, 0, 0), -transform.up, hit1, -
    rayLength, layerMask);
```

Výpis 2: Část kodu - generuje paprsek



Obrázek 4: Raycasting u auta umělé inteligence

6.3 Tramvaj

I přesto, že tramvaje, stejně jako auta umělé inteligence, jezdí po předem vyznačené trase, nejde o stejný systém. Zatímco auta umělé inteligence jsou řízená systémem směrových bodů, pohyb tramvají je předem vytvořená animace po křivce. Tramvaj tak nelze vychýlit z její trasy. To ovšem neznamená, že tramvaje nereagují na určité situace.

Ovládací skript tramvají zajišťuje plynulý rozjezd a zpomalování pomocí změny rychlosti animace v předem určených časových inkrementech 3 . Unity automaticky vyhlazuje animaci a tím pádem nedochází k sekanému pohybu. Tento skript rovněž ovládá zastavování tramvaje v určitých situacích. Pro tento účel jsou využity dvě metody. Raycasting se používá pro zastavení na místech, kde se nedá předem určit, zda bude muset tramvaj zastavit. Těmi jsou světelné křižovatky a zastavení za ostatními tramvajemi. Pro zpomalování nebo zastavování na zastávkách jsou využity triggery. Pokud jimi tramvaj projede, odešle se do ovládacího skriptu informace z triggeru a tramvaj buď zpomalí na určenou rychlost, nebo úplně zastaví. Skript rovněž umí zastavit na předem určenou dobu. Této funkcionality bylo využito pro zastavování na zastávkách.

```
if (decelerate)
{
    if (speed >= decelerateValue)
    {
        if (timeExceeded)
        {
            for (var state : AnimationState in anim)
            {
                state.speed -= 0.05;
                speed -= 0.05;
            }

            timeExceeded = false;
        }
    }
    else
    {
        decelerate = false;
        decelerateValue = 0.0;
    }
}

if (decelerate || accelerate)
{
    timer -= Time.deltaTime;

    if (timer < 0.0)
    {
        timeExceeded = true;
        timer = 0.1;
    }
}
```

Výpis 3: Kód pro plynulé zpomalení animace

6.4 Detekování dopravních přestupků

Pro správné detekování jednotlivých dopravních přestupků, musely být vytvořeny i jednotlivé spouštěcí objekty. Ty komunikují s právě aktivním (kolidujícím) objektem hráče, kde se poté vyhodnotí, zda byl porušen některý z dopravních předpisů.

6.4.1 Detekce rychlosti

Jedním z nejzákladnější dopravních předpisů, které bylo v této aplikaci nutné hlídat, je rychlost hráče. Samotné vyhodnocení je vyřešeno jednoduchou podmínkou porovnávající aktuální rychlost s rychlostním omezením. To se mění podle zóny, ve které se právě nacházíme. Změna je zapříčiněná projetím triggeru s připojeným skriptem, který má přiřazené požadované rychlostní omezení. Samotná změna nastane až po ukončení

kontaktu s triggrem. Pro vyznačení hranice rychlostní zóny je tedy třeba dvou triggerů za sebou, kde každý z nich mění rychlost na tu stranu, na které se nachází.

Tento problém by se dalo vyřešit i zavedením velkých trigger zón, kde by se konstantně kontrolovalo, v jaké zóně se hráč nachází. Zatímco je tato metoda jednodušší z hlediska návrhu úrovně, oproti použité metodě je daleko více výkonově náročná.

6.4.2 Detekce stopky

Detekování správného projetí stopky je o něco složitější než se může zdát. Je totiž třeba rozlišovat, z které strany auto přijíždí. Z toho důvodu je detekce stop vytvořena pomocí dvou triggerů. Ty nastavují stav projetí stopkou, podle toho, v jakém pořadí jimi hráč projíždí. Zároveň se v konkrétním stavu detekuje, zda hráč zastavil automobil. Pokud se u hráče detekuje zastavení v konkrétním stavu, je stopka detekována jako projetá správně, pokud ale hráč zastaví příliš brzy nebo příliš pozdě, je stopka vyhodnocená jako nesprávně projetá. Pokud hráč projede stopkou z opačné strany, detekuje se projetí jako správné. Důležité je i vrácení stavu do původní podoby, pokud by hráč projetí stopkou nedokončil, například by vjel pouze do prvního triggeru, ale poté by vycouval.

6.4.3 Železniční přejezd

Při detekování správného projetí železničního přejezdu se kontroluje hned několik parametrů. První z nich je, zda hráč vjel na přejezd, když blikaly červená světla. Druhým parametrem je rychlost, jakou může hráč projet železniční přejezd. Jelikož detekce rychlosti byla již vyřešená, bylo zbytečné implementovat ji přímo do železničního přejezdu. Místo toho se jednoduše před železničním přejezdem změní omezení rychlosti a za přejezdem se zase vrátí na původní hodnotu.

Pro samotnou detekci vjezdu na železniční přejezd byl vytvořen automatizovaný systém reálně reagující na přijíždějící vlak. Nebylo použité žádné časově přesné skriptování. Proto je tento systém nezávislý na počtu kolejí a počtu jezdících vlaků.

Systém funguje podobně jako v reálném světě. Přijíždějící vlak je od určité vzdálenosti zachycen triggerem. Tím se spustí červená výstražná světla na přejezdu a začne se detekovat, zda hráč nevjel do kolejiště. Rozlišit, zda vjel do triggeru vlak nebo hráč, bylo možné pomocí tagů. Jakmile vlak opustí trigger, celý systém se přepne zpět do výchozího stavu.

6.4.4 Jednosměrná ulice

Pro tvorbu jednosměrné ulice byla použita podobná metoda jako u stopky. Stejně jako tam bylo zapotřebí zjistit, z jaké strany jednosměrné ulice auto vjíždí a zároveň udržovat vždy správný stav. V tom je ale jedna podstatná změna. Jakmile se dostane hráč za vstupní bod jednosměrné ulice, musí ji dokončit. Vyjetí vstupním bodem by znamenalo, že hráč změnil směr jízdy nebo začal couvat. Ani jedno není v jednosměrné ulici dovoleno (couvání pouze pro parkování).

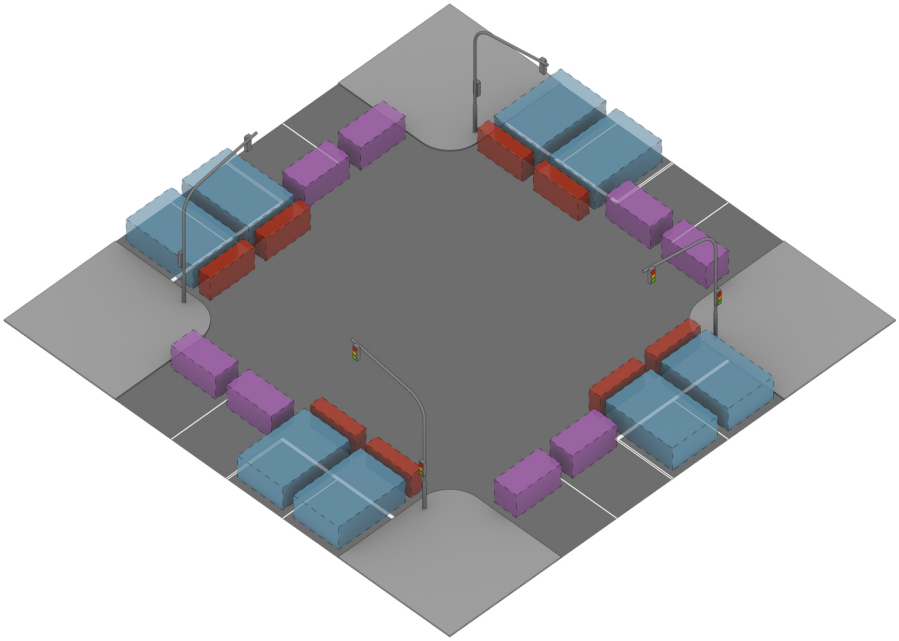
Technicky je jednosměrná ulice řešená čtyřmi triggery, které přiřazují stavy závislé na konkrétním stavu. Správná posloupnost stavů znamená správné projetí jednosměrné ulice.

6.4.5 Světelná křižovatka

Systém semaforů je jeden ze složitějších systémů vytvořených pro tuto aplikaci. Základními požadavky byla správná změna stavu semaforu, detekce jízdy na červenou, závislá na stavu semaforu a detekce jízdy ve správném pruhu. Kromě toho musely být semafory synchronizovány pro síťovou hru.

Původní návrh spojoval všechny tyto mechanismy do jednoho prefabu. Tento postup se ale ukázal jako velmi nepraktický, jelikož byla úprava systému i samotného prefabu složitá a často vedla k neočekávaným chybám.

Proto byl tento systém rozdělen na několik částí 5. První část je skript, který má na starost přepínání stavů. Tomuto skriptu se v inspectoru nastaví jednotlivé stavy a časy pro tyto stavy. Ten pak prochází pole stavů a přepíná je podle zadaných časů. Tento skript se poté přiřadí jednotlivým semaforům, které jsou ovládány podle aktuálního stavu. Prefab semaforu obsahuje kromě grafiky semaforu, světel a triggeru pro detekci jízdy na červenou také skript pro ovládání semaforu. Tomu se přiřadí světla semaforu, trigger pro detekci jízdy na červenou a směr jízdy. Podle směru jízdy a aktuálního stavu se pak rozsvěcují světla semaforu. Zároveň se zapíná a vypíná trigger pro detekci jízdy na červenou. Pokud je na semaforu červená, trigger je zapnutý a detekuje případné projetí hráčem. Poslední částí je detekce jízdy ve správném pruhu. Tato část je nezávislá na ostatních a lze ji použít i mimo semafor. Využívá vstupní a výstupní triggery. Vstupní trigger je umístěný v pruzích mířících do křižovatky a výstupní trigger v pruzích mířících z křižovatky ven. Každý vstupní trigger má pak přiřazen jeden nebo více výstupních triggerů. Hráči projíždějícímu křižovatkou je po kolizi se vstupní triggerem přiřazeno jeden nebo více výstupních triggerů. Pokud projede jedním z nich, je projetí křižovatky vyhodnoceno jako správné. V opačné případě se detekuje přestupek.



Obrázek 5: Rozložení triggerů u světelné křižovatky.

7 Síťová hra

Rozdíl mezi hrou pro jednoho hráče a hrou více hráčů je na první pohled ne příliš znatelný, ovšem z vývojářského hlediska je to velká výzva. Je třeba si uvědomit které objekty je nutno synchronizovat a jakým způsobem je synchronizovat. Zároveň je třeba se zamyslet nad tím, které operace se mají provádět na straně klienta a které na straně serveru.

Unity nabízí dva způsoby synchronizace:

- Remote Procedure Calls – používají se pro spuštění funkce u ostatních účastníků připojených do sítě. Klient odesílá RPC na server a server odesílá RPC jednomu nebo více klientům. Používá se pro nepravidelnou synchronizaci dat. Často se jedná o synchronizaci vyvolanou akcí hráče. Například rozsvícení brzdových světel, když hráč začne brzdit. Výhodou RPC je malý počet dat posílaný sítí.
- State Synchronization – používá se pro synchronizaci konstantně se měnících dat. Typickým příkladem je pozice auta, které neustále jezdí a proto je zapotřebí neustále synchronizovat jeho pozici. Nevýhodou této metody je velký počet dat posílaných přes síť. Unity ovšem disponuje módem přenosu Reliable Delta Compressed. Ten umožnuje přenášet pouze ta data, která se změnila.

Při tvorbě síťové hry, je zapotřebí určit, která část kódu se má vykonat na straně klienta a které na straně serveru. To je možno zjistit pomocí třídy Network. Tato třída umožňuje získat různé informace jako například celkový počet připojených hráčů, lokální NetworkPlayer instanci, zdali se jedná o server nebo klient, a mnoho dalších.

Často je zapotřebí určit i vlastnictví instance a vykonat kód pouze u konkrétního účastníka. Toho je možné dosáhnout pomocí přístupu ke komponentě networkView a proměnné isMine. Toto opatření je klíčové například pro ovládání auta. Pokud se připojí více hráčů, je vytvořeno více instancí stejného prefabu a je proto nutné určit, kdo ovládá které auto. Zároveň je networkView použito pro volání RPC.

V této aplikaci byl pro vytvoření serveru využit Master Server 4 , což je zprostředkovatel spojení mezi jednotlivými klienty. Pro použití Master serveru je zapotřebí zaregistrovat hru. Po vytvoření serveru a zaregistrování hry je možné na straně klienta tuto konkrétní hru vyhledat a připojit se.

```
function StartServer()
{
    Network.InitializeServer(16, 25001, !Network.HavePublicAddress);
    MasterServer.RegisterHost(gameName, "BP Network Test", "BP Network Test");
}
```

Výpis 4: Registrace serveru

Při připojení je nejdříve nutno vyhledat konkrétní hru. Třída MasterServer obsahuje funkci RequestHostList 5 , která vyhledá hru podle unikátního názvu hry. Pro zpřehlednění lze pomocí jednoduché podmínky indikovat hledání, dokud se nenajde nějaký výsledek 6 . Tak je možno získat hostData pro hru, ke které se chce klient připojit. HostData

obsahují informaci o serveru, kde se chce klient připojit. Je tak možno získat IP adresu serveru, počet připojených hráčů, komentář ke hře, atd. Připojení pak probíhá pomocí metody Connect třídy Network 7 . Té se zadává jako parametr konkrétní hostData.

```
function refrehsHostList()
{
    MasterServer.RequestHostList(gameName);
    refreshing = true;
}
```

Výpis 5: Funkce pro získání hostData

```
if (refreshing)
{
    if (MasterServer.PollHostList().Length > 0)
    {
        refreshing = false;
        Debug.Log(MasterServer.PollHostList().Length);
        hostData = MasterServer.PollHostList();
    }
}
```

Výpis 6: Podmínká pro ověření refreshing

```
if (GUI.Button(Rect(240, 20 + (34 * i), 340, 30), hostData[i].gameName))
{
    gameStarterObject.SendMessage("ChangeGameState", 2);
    Network.Connect(hostData[i]);
}
```

Výpis 7: Tlačítko pro připojení do hry

8 Grafická optimalizace

Vytvářet hru s otevřeným světem je výzva hlavně co se týče vykreslování grafiky. V případě této aplikace, kdy má mapa přes tři čtverečné kilometry, bylo zapotřebí využít různých nástrojů a triků, aby byla hratelná na průměrném počítači.

8.1 Omezení počtu draw callů

Jedním z hlavních způsobů optimalizace je omezení počtu draw callů. Počet draw callů je závislý na počtu objektů, počtu materiálů a nasvícení ve scéně. Scéna s jedním objektem, který má jeden materiál a je nasvícený světlem, které má vypnuté stíny, bude mít jeden draw call. Stačí ale, když se zapnou ostré stíny a počet draw callů se zvýší na tři. Každý další přidaný objekt nebo materiál přidá několik dalších draw callů.

Unity disponuje metodou pro efektivní automatizované redukovaní drawcallů. Tato metoda se jmenuje Batching a redukuje draw cally u objektů se stejným materiálem. Existuje dynamic batching a static batching. Jak již názvy napovídají, první jmenovaný se týká nestatických objektů, zatímco ten druhý statických. Dynamický batching má několik omezení a nevýhod. Jednou z hlavních je omezení počtu vertexů na 900 pro jeden objekt. Statický batching nemá omezení složitosti geometrie a vyžaduje méně CPU výkonu, ovšem je třeba si uvědomit, že se dá použít pouze pro statické objekty.

Další metodou, nad kterou má vývojář plnou kontrolu, je manuální úprava assetů pro optimalizaci draw callů. Obecně platí, že pokud máme mnoho objektů se stejným materiálem, které mohou být sloučeny, je dobré je sloučit. Příkladem mohou být kameny u okraje silnice. Je zbytečné vykreslovat každý kámen zvlášť, když mají všechny stejný materiál. Po sloučení je možno ušetřit řádově až desítky draw callů. Samotné sloučení lze provést ve 3D grafickém editoru, nebo pomocí skriptu přímo v Unity.

Průměrný počet draw callů, který je na počítačích střední třídy přípustný, se pohybuje mezi 1800 až 2500 draw cally. Tato aplikace má průměrně 600 draw callů, což je částečně způsobeno nevelkým počtem unikátních modelů, ale zároveň i dobrou optimalizací.

8.2 Level of detail (LOD)

Další velmi populární technikou, pro optimalizaci vykreslování je používání tzv. LODů. Hlavním principem LOD je měnit kvalitu 3D modelů podle předem určených parametrů (vzdálenost, důležitost ve scéně, atd.). Rozlišujeme dva druhy LOD :

- Statický LOD – při tomto způsobu musíme vytvořit více verzí daného 3D modelu, přičemž každá verze má jinou úrveň detailu. Výhodou je nenáročné řešení za běhu programu, kdy se pouze přepínají různé úrovně detailu. Nevýhodou je „blikání“, kdy je často vidět moment přepnutí mezi jednotlivými úrovněmi detailu.
- Dynamický LOD – samotná degradace modelu probíhá za běhu programu. Výhodou je plynulejší přepínání mezi jednotlivými úrovněmi detailu. Nevýhodou jsou vyšší nároky na výpočet. Zároveň se nedá použít v každé situaci – u některých

3D modelů dosáhneme lepších výsledků ruční degradací modelu v 3D modelovací aplikaci a následným použitím statického LOD.

V této aplikaci byly použity LODy primárně na budovy. Pro každou budovy byly vytvořeny dva 3D modely. Jeden s vysokými detaily a druhý velmi zjednodušený. Důležité je zachovat stejnou velikost, tvar budovy a mapování textur. Výsledné modely byly naimportovány do Unity a pomocí komponenty LOD Group, připojené na prázdný game object, se nastavily vzdálenosti zobrazování jednotlivých verzí budov. Tento výsledný objekt byl poté uložen jako prefab, pro jednodušší použití ve scéně.

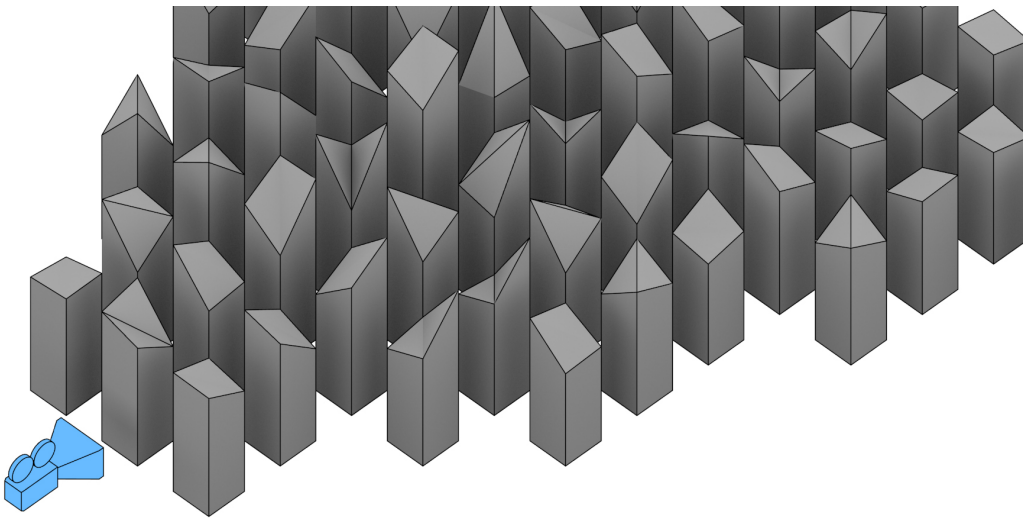
8.3 Occlusion culling

Unity používá automaticky frustum culling 6 . To dovoluje určit ořezání obrazu pomocí near clip plane a far clip plane. Zároveň se nevykreslují objekty, které leží mimo výseč kamery. Tento typ vykreslování je ovšem velmi nevhodný, jelikož se vykresluje obrovské množství objektů, které hráč vůbec nevidí.

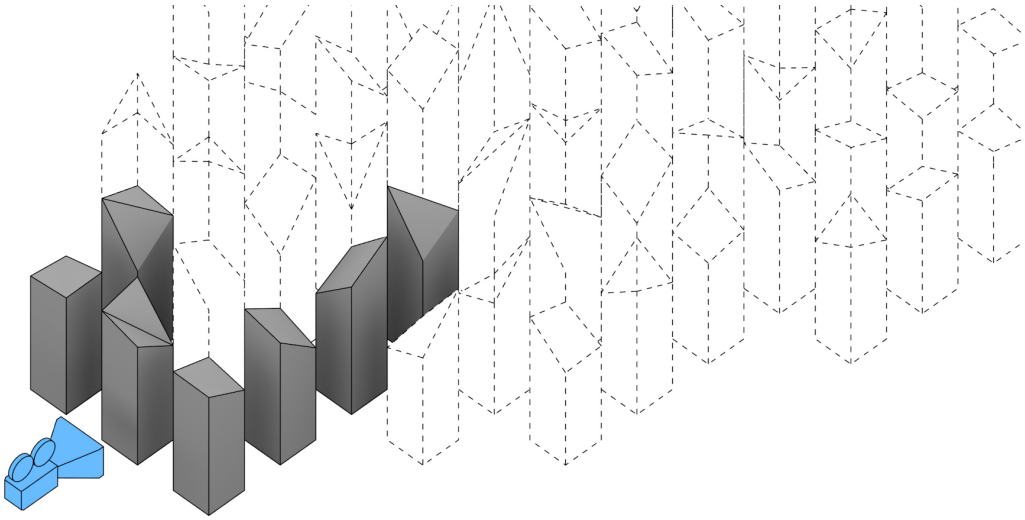
Proto se používá právě occlusion culling 7 . To dovoluje určit viditelnost objektů a vykreslit pouze ty, které jsou reálně vidět. Unity disponuje třemi způsoby výpočtu occlusion culling:

- PVS Only – occlusion culling je vypočteno pouze pro statické objekty. Dynamické objekty jsou ovlivněné pouze frustum cullingem. Tato varianta je nejméně náročná na CPU, jelikož je viditelnost objektů vypočítána předem. Zároveň není vhodná pro použití ve scéně s mnoha dynamickými objekty.
- PVS and dynamic objects – zde je viditelnost statických objektů vypočítána předem a dynamické objekty využívají portal culling. Jelikož je viditelnost vypočítána předem, nemohou se portály otevírat a zavírat za běhu aplikace. Tato varianta je kompromisem mezi přesností a využitým výkonem.
- Automatic Portal Generation – zde jsou portály generovány automaticky a jsou použity pro výpočet viditelnosti jak statických, tak i dynamických objektů. Portály je možné otevírat a zavírat za běhu aplikace. Tato varianta occlusion cullingu je nejpřesnější, ale zároveň je nejnáročnější na CPU.

V této bakalářské práci byla využita metoda PVS and dynamic objects. Důvodem byla nutnost aplikovat occlusion culling i na dynamické objekty s co nejmenším využitím CPU. Zároveň mapa neobsahuje mnoho objektů s velkým počtem polygonů a tak i při nepřesnostech occlusion cullingu se počet vykreslovaných polygonů o mnoho nenavýší.



Obrázek 6: Frustum culling.



Obrázek 7: Occlusion culling.

9 Uživatelské rozhraní (GUI)

Unity nedisponuje žádným nástrojem pro jednoduchou tvorbu GUI. Veškeré elementy uživatelského rozhraní je tedy nutné tvořit kódem 8. To sice není ideální z designového hlediska, ale na druhou stranu tím vývojář dostane stoprocentní kontrolu na uživatelským rozhraním.

Pro tuto aplikaci byl vzhled GUI změněn. Změny vzhledu GUI lze dosáhnou vytvořením vlastního GUI Skinu. Pomocí GUI Skinu lze změnit parametry všech elementů. Pro použití GUI Skinu je nutné ho přiřadit ve funkci pro vykreslení GUI, v opačném případě bude použit základní GUI Skin.

Často je zapotřebí změnit vzhled nějakého prvku GUI jen v konkrétním případě. Pro tento účel lze využít GUI Style. Pomocí GUI Style lze nadefinovat jednotlivé změny vzhledu a poté je aplikovat pouze na konkrétní element GUI.

Unity sice nemá žádný editor GUI, je však možné takový nástroj zakoupit na Asset storu. Nejpopulárnějším z nich je NGUI, který je nyní upravován a bude obsažen v jedné z dalších verzí Unity.

```
GUI.BeginGroup(Rect((Screen.width / 2) - 100, (Screen.height / 2) - 100, 200, 200));

GUI.Box(Rect(0, 0, 200, 200), "Menu");

if (GUI.Button(Rect(0, 50, 200, 50),  "Resume"))
{
    inGameMenu01 = false;
}
if (GUI.Button(Rect(0, 100, 200, 50), "Main Menu"))
{

    Network.RemoveRPCs(Network.player);
    Network.DestroyPlayerObjects(Network.player);
    Network.Disconnect();
    Application.LoadLevel(0);

}
if (GUI.Button(Rect(0, 150, 200, 50), "Quit"))
{
    Network.RemoveRPCs(Network.player);
    Network.DestroyPlayerObjects(Network.player);
    Network.Disconnect();
    Application.Quit();
}

GUI.EndGroup();
```

Výpis 8: Část kodu - GUI

10 Závěr

Cílem této práce bylo seznámit se s možnostmi a funkcemi herního enginu Unity 3D a vytvořit ukázkovou aplikaci v podobě virtuální autoškoly. Tato práce obsahuje rozbor a srovnání konkurenčních enginů. Dále se zabývá samotným Unity a popisuje základní nástroje a funkce, které jsou později použity při tvorbě aplikace. Zároveň byl popsán proces tvorby grafiky, její úpravy pro herní engine, exportu do správného formátu a nastavení importu v Unity.

Pro praktickou demonstraci funkčnosti Unity byla vytvořena aplikace virtuální autoškola. V této aplikaci bylo použito mnoho nástrojů a funkcí popsaných v této bakalářské práci. Při tvorbě se vyskytly problémy s přílišnou náročností. Ty byly vyřešeny upravením mapy a aplikováním occlusion cullingu. Vývoj síťové hry byl jedním z nejtěžších úkolů. Velkým přínosem byla možnost průběžně testovat aplikaci v kolektivu ostatních studentů. Bylo tak možné odhalit chyby, které nejsou z vývojářského pohledu hned zřejmé. Celý proces byl pečlivě zdokumentován včetně ilustračních obrázků a ukázek kódu.

Při tvorbě virtuální autoškoly byly zaznamenány i nedostatky Unity. Jedním z největších nedostatků je nekompletnost Unity. Oproti ostatním herním enginům Unity postrádá některé klíčové nástroje. Tento problém je možné vyřešit zakoupením konkrétních nástrojů na Asset Storu. Dalším nedostatkem je editor scény. Ten je oproti ostatním enginům velmi zjednodušený a postrádá některé pokročilejší funkce, které by urychlily vývoj.

Aplikace virtuální autoškoly by se dala dále rozvíjet v mnoha směrech. Jelikož se jedná o aplikaci s otevřeným světem, bylo by možné tento svět zvětšit. Systémově by se dala tato aplikace vylepšit například přidáním chodců, kteří by reagovali na provoz nebo vylepšením umělé inteligence.

11 Reference

[1] SCHRÖTER, Zdeněk, *Autoškola? Pohodlně!,* aktualiz. vyd. Plzeň: Agentura Schröter, 2013, 288 s. ISBN 978-80-904665-9-3.

[2] *Unity Manual* [online], Unity Technologies, c2014, https://docs.unity3d.com/Documentation/Manual/index.html.

[3] *Unity Script Reference* [online], Unity Technologies, c2014, https://docs.unity3d.com/Documentation/ScriptReference/index.html.

[4] *Unity Community Wiki* [online], Unify Community, c2014, http://wiki.unity3d.com/index.php/.

[5] *Unity Cookie* [online], CG Cookie, Inc, c2014, http://cgcookie.com/unity/.

[6] *Polycount Wiki* [online], polycount.com , c2014, http://wiki.polycount.com/.

[7] *3D Motive* [online], 3D Motive LLC , c2014, https://www.3dmotive.com/.

Externí materiály

Pro tuto práci byly použity materiály z externích zdrojů. Veškeré tyto materiály jsou distribuovány pod licencí Royalty Free nebo Creative Commons.

Název	Typ	Autor	Web
Car 3D Model	3D model	Matvei	archive3d.com
PickUp	3D Model	3dregenerator	tf3dm.com
Van Kendo	3D Model	3dregenerator	tf3dm.com
Locomotive TE7	3D Model	3dregenerator	tf3dm.com
Car Horn Honk 1	Zvuk	SoundBible	SoundBible.com
Terrain Assets	3D model	Unity Technologies	assetstore.unity3d.com
Textury	textury	CGTextures	cgtextures.com